Govt. College University, Lahore

Assignment # 01

Submitted by:

Name: M. Umar Farooq

Roll No. 6320-BSAI-23

Submitted to:

Mam Qurra tul Ann Sikander

Subject: Data Structures and Algorithms

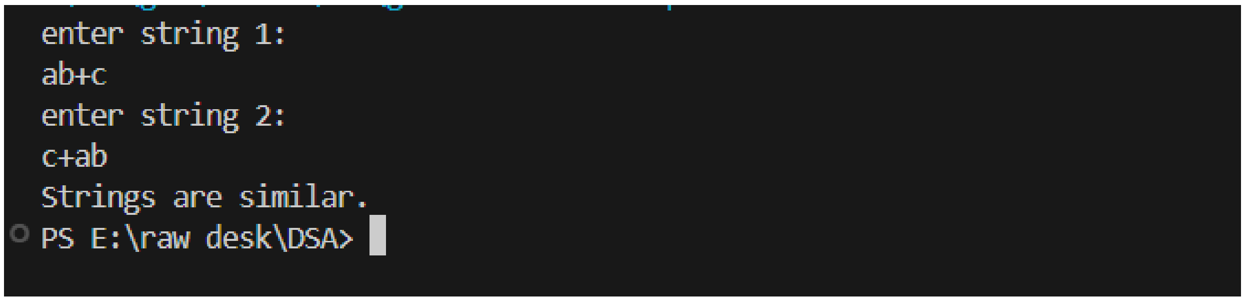
Department of Artificial Intelligence

Question # 01 (Similar strings)

Code:

```
#include <iostream>
#include <stack>
using namespace std;

bool areStringsSimilar(string str1, string str2) {
    if (str1.length() != str2.length()){
         return false;
      }
    stack<char> stack1, stack2;

    for (int i = 0; i < str1.length(); i++) stack1.push(str1[i]);
    for (int i = 0; i < str2.length(); i++) stack2.push(str2[i]);

    while (!stack1.empty()) {
        char ch = stack1.top();
        stack1.pop();

        stack<char> tempStack;
        bool found = false;

        while (!stack2.empty()) {
            if (stack2.top() == ch && !found) {
                stack2.pop();
                found = true;
            } else {
                tempStack.push(stack2.top());
                stack2.pop();
            }
        }

        while (!tempStack.empty()) {
            stack2.push(tempStack.top());
            tempStack.pop();
        }

        if (!found) return false;
    }

    return stack2.empty();
}

int main() {
    string s1, s2;
    cout<<"enter string 1: "<<endl;
    cin>>s1;
    cout<<"enter string 2: "<<endl;
    cin>>s2;

    if (areStringsSimilar(s1, s2))
        cout << "Strings are similar.\n";
    else
        cout << "Strings are not similar.\n";

    return 0;
}
```

Output:



```
enter string 1:
ab+c
enter string 2:
c+ab
Strings are similar.
PS E:\raw desk\DSA>
```

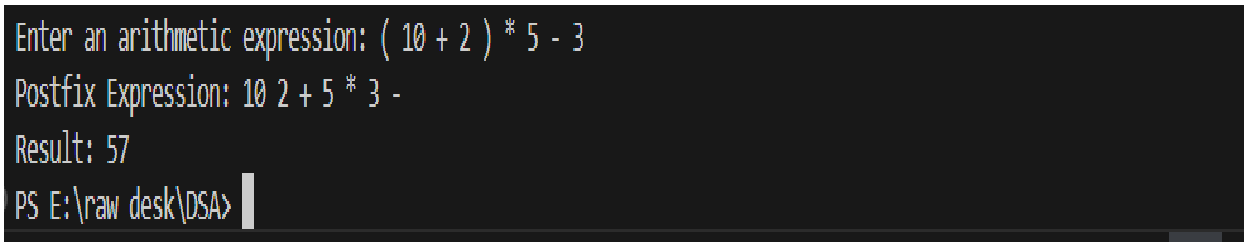
Question # 02 (Reverse Polish Notation)

Code:

```
#include <iostream>
#include <stack>
using namespace std;

int precedence(char op) {
    if (op == '+' || op == '-') return 1;
    if (op == '*' || op == '/') return 2;
    if (op == '^') return 3;
    return 0;
}

bool isOperator(char ch) {
```

```cpp
    return ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == '^';
}

string infixToPostfix(string infix) {
    stack<char> operators;
    string postfix = "";
    string token = "";

    for (size_t i = 0; i < infix.length(); i++) {
        char ch = infix[i];
        if (isdigit(ch)) {
            token += ch;
        } else {
            if (!token.empty()) {
                postfix += token + " ";
                token = "";
            }
            if (ch == '(') {
                operators.push(ch);
            } else if (ch == ')') {
                while (!operators.empty() && operators.top() != '(') {
                    postfix += operators.top(); postfix += " ";
                    operators.pop();
                }
                if (operators.empty()) throw runtime_error("Unbalanced parentheses");
                operators.pop();
            } else if (isOperator(ch)) {
                while (!operators.empty() && precedence(operators.top()) >= precedence(ch)) {
                    postfix += operators.top(); postfix += " ";
                    operators.pop();
                }
                operators.push(ch);
            } else if (ch != ' ') {
                throw runtime_error("Invalid token in expression");
            }
        }
    }
    if (!token.empty()) postfix += token + " ";
    while (!operators.empty()) {
        if (operators.top() == '(') throw runtime_error("Unbalanced parentheses");
        postfix += operators.top(); postfix += " ";
        operators.pop();
    }
    return postfix;
}

int evaluatePostfix(string postfix) {
    stack<int> values;
    string token = "";

    for (size_t i = 0; i < postfix.length(); i++) {
        char ch = postfix[i];
        if (isdigit(ch)) {
            token += ch;
        } else {
            if (!token.empty()) {
                values.push(stoi(token));
                token = "";
            }
            if (isOperator(ch)) {
                if (values.size() < 2) throw runtime_error("Invalid expression");
                int b = values.top(); values.pop();
                int a = values.top(); values.pop();

                if (ch == '+') values.push(a + b);
                else if (ch == '-') values.push(a - b);
                else if (ch == '*') values.push(a * b);
                else if (ch == '/') {
                    if (b == 0) throw runtime_error("Division by zero");
                    values.push(a / b);
                } else if (ch == '^') {
                    int result = 1;
                    for (int j = 0; j < b; j++) result *= a;
                    values.push(result);
                }
            }
        }
    }
    if (values.size() != 1) throw runtime_error("Invalid expression");
    return values.top();
}

int main() {
    try {
        string infix;
        cout << "Enter an arithmetic expression: ";
        getline(cin, infix);

        string postfix = infixToPostfix(infix);
        cout << "Postfix Expression: " << postfix << endl;

        int result = evaluatePostfix(postfix);
        cout << "Result: " << result << endl;
    } catch (runtime_error e) {
        cout << "Error: " << e.what() << endl;
    }
    return 0;
}
```

Output:



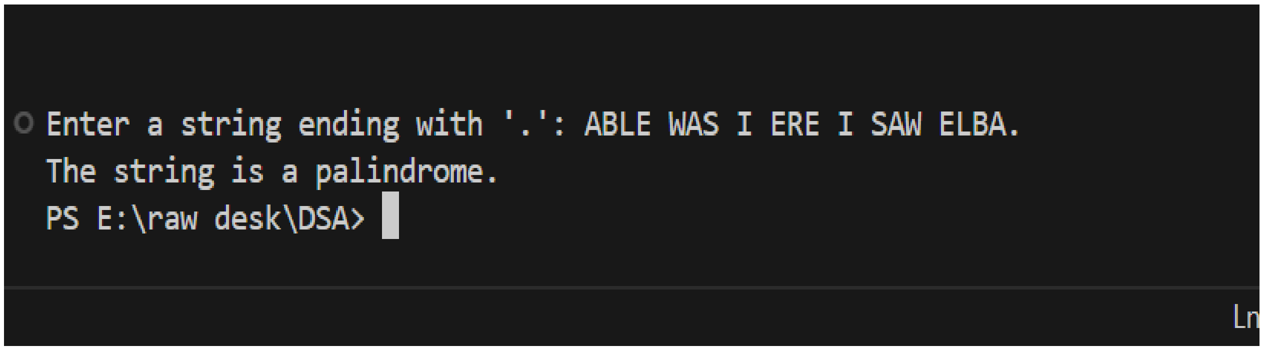
```
Enter an arithmetic expression: ( 10 + 2 ) * 5 - 3
Postfix Expression: 10 2 + 5 * 3 -
Result: 57
PS E:\raw desk\DSA>
```

Question # 03 (Palindrome)

Code:

```
#include <iostream>
#include <stack>

using namespace std;

bool isPalindrome(string str) {
    stack<char> s;
    string filtered = "";

    for (int i = 0; i < str.length(); i++) {
        if ((str[i] >= 'A' && str[i] <= 'Z') || (str[i] >= 'a' && str[i] <= 'z')) {
            char lowerCh = (str[i] >= 'A' && str[i] <= 'Z') ? str[i] + 32 : str[i];
            s.push(lowerCh);
            filtered += lowerCh;
        }
    }
    while (!s.empty()) {
        if (s.top() != filtered[filtered.length() - s.size()]) {
            return false;
        }
        s.pop();
    }
    return true;
}

int main() {
    string str;
    cout << "Enter a string ending with '.': ";
    getline(cin , str);

    if (!str.empty() && str.back() == '.') {
        str.pop_back();
    }

    if (isPalindrome(str)) {
        cout << "The string is a palindrome." << endl;
    } else {
        cout << "The string is not a palindrome." << endl;
    }

    return 0;
}
```

Output:



```
Enter a string ending with '.': ABLE WAS I ERE I SAW ELBA.
The string is a palindrome.
PS E:\raw desk\DSA>
```